Projet Big Data

Module Clôture

Détection de Fraude Bancaire

avec Apache Spark et Machine Learning

Technologies utilisées :

Apache Spark • Spark SQL • MLlib • Spark Streaming • Grafana

Équipe :

Ahmed Dinari • Bilel Samaali • Anas Belhouichet

Date : 13 janvier 2026

Module : Big Data Hadoop

Contributions de l'Équipe

Membre	Rôle	Contributions
Ahmed Dinari	Lead Developer & ML Engineer	Architecture pipeline, Spark Core, Modèles MLlib, Configuration Docker
Bilel Samaali	Data Engineer	Spark SQL Analytics, Nettoyage données, Feature Engineering
Anas Belhouichet	Visualisation & Documentation	Dashboard Grafana, Rapport LaTeX, Présentation Beamer

TABLE 1 – Répartition du travail

Table des matières

Contributions de l'Équipe **1**

1 Introduction **3**
1.1 Contexte du Projet . . . 3
1.2 Objectifs . . . 3
1.3 Dataset . . . 3

2 Architecture du Pipeline **3**
2.1 Vue d'Ensemble . . . 3
2.2 Composants . . . 4
2.3 Flux de Données . . . 4

3 Traitement avec Spark SQL **4**
3.1 Nettoyage des Données . . . 4
3.2 KPIs Calculés . . . 5
3.3 Analyse par Tranche Horaire . . . 5

4 Machine Learning avec MLlib **5**
4.1 Préparation des Features . . . 5
4.2 Modèles Entraînés . . . 5
4.2.1 RandomForest Classifier . . . 5
4.2.2 Logistic Regression . . . 6
4.3 Résultats . . . 6
4.4 Matrice de Confusion (RandomForest) . . . 6
4.5 Importance des Features . . . 6

5 Spark Streaming **6**
5.1 Architecture Streaming . . . 7
5.2 Flux de Traitement . . . 7
5.3 Métriques Streaming . . . 7

6 Dashboard Grafana **7**
6.1 Panels Implémentés 7
6.2 Configuration 7

7 Analyse de Graphes avec GraphX **8**
7.1 Objectif 8
7.2 Implémentation 8
7.3 Résultats GraphX 8
7.3.1 Communautés Détectées 8
7.3.2 Top Features par Séparation 9

8 Federated Learning (Concept) **9**
8.1 Architecture Proposée 9
8.2 Avantages 9
8.3 Protocole FedAvg 9

9 Déploiement Azure Cloud **10**
9.1 Architecture Azure 10
9.2 Composants Azure 10
9.3 Configuration Databricks 10
9.4 Captures Azure Portal 11

10 Visualisations et Résultats **11**
10.1 Comparaison des Modèles 11
10.2 Importance des Features 12
10.3 Courbe ROC 13
10.4 Distribution des Classes 13
10.5 Distribution Horaire 14
10.6 Interface Spark UI 14
10.7 MLlib et Modélisation 16
10.8 Dashboard Grafana 17

11 Conclusion **18**
11.1 Réalisations 18
11.2 Difficultés Rencontrées 18
11.3 Améliorations Futures 18

12 Références **18**

A Structure du Projet **19**

B Commandes d'Exécution **19**

1 Introduction

1.1 Contexte du Projet

La détection de fraude bancaire est un enjeu majeur dans le secteur financier. Avec des millions de transactions effectuées quotidiennement, il est crucial de pouvoir identifier rapidement les transactions frauduleuses tout en minimisant les faux positifs.

Ce projet met en œuvre un pipeline Big Data complet pour la détection de fraude en temps réel, utilisant l'écosystème Apache Spark.

1.2 Objectifs

— **Ingestion et nettoyage** des données avec Spark SQL
— **Analyse exploratoire** avec des KPIs pertinents
— **Machine Learning** avec MLlib (RandomForest, Logistic Regression)
— **Streaming temps réel** pour la détection continue
— **Visualisation** via dashboard Grafana

1.3 Dataset

Credit Card Fraud Detection Dataset (Kaggle)
— **Source :** `https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud`
— **Transactions :** 284,807
— **Features :** 30 (V1-V28 transformées PCA + Time + Amount)
— **Classe :** Binaire (0 = Normal, 1 = Fraude)
— **Distribution :** 99.83% Normal, 0.17% Fraude

2 Architecture du Pipeline

2.1 Vue d'Ensemble

ARCHITECTURE DU PIPELINE BIG DATA

`CSV Data` → `Spark Core` → `Spark SQL` → `MLlib` → `Streaming` → `Grafana`

FIGURE 1 – Pipeline de traitement des données

2.2 Composants

Composant	Technologie	Rôle
Ingestion	Spark Core	Chargement CSV
Nettoyage	Spark SQL	Transformation, filtrage
Analytics	Spark SQL	KPIs, agrégations
ML Model	MLlib	Classification fraude
Streaming	Structured Streaming	Temps réel
Visualisation	Grafana	Dashboard interactif

TABLE 2 – Composants du pipeline

2.3 Flux de Données

1. **Extraction :** Chargement du fichier CSV (284K transactions)
2. **Transformation :** Nettoyage, feature engineering, normalisation
3. **Analyse SQL :** Calcul des KPIs métier
4. **Modélisation :** Entraînement RandomForest et Logistic Regression
5. **Évaluation :** Métriques de performance (AUC, Precision, Recall)
6. **Streaming :** Simulation de flux temps réel
7. **Export :** Fichiers Parquet/JSON pour Grafana

3 Traitement avec Spark SQL

3.1 Nettoyage des Données

Listing 1 – Nettoyage avec Spark SQL

```
# Suppression des valeurs nulles
df_clean = df.dropna()

# Filtrage des montants invalides
df_clean = df_clean.filter(col("Amount") > 0)

# Ajout de features derivees
df_clean = df_clean.withColumn(
    "Hour", (col("Time") / 3600).cast("integer") % 24
).withColumn(
    "Is_High_Amount", when(col("Amount") > 500, 1).otherwise(0)
)
```

3.2 KPIs Calculés

Métrique	Valeur
Total Transactions	282,982
Transactions Fraude	465
Taux de Fraude	0.1643%
Montant Moyen	$88.92
Montant Max	$25,691.16
Montant Min	$0.01
Écart-type	$250.82

TABLE 3 – Statistiques globales du dataset (après nettoyage)

3.3 Analyse par Tranche Horaire

Les transactions sont analysées par heure pour identifier les patterns temporels de fraude. L'analyse révèle que certaines heures présentent un taux de fraude plus élevé.

4 Machine Learning avec MLlib

4.1 Préparation des Features

Listing 2 – Feature Engineering

```
# Assemblage des features
assembler = VectorAssembler(
    inputCols=["V1", "V2", ..., "V28", "Amount"],
    outputCol="features_raw"
)

# Normalisation
scaler = StandardScaler(
    inputCol="features_raw",
    outputCol="features",
    withStd=True, withMean=True
)
```

4.2 Modèles Entraînés

4.2.1 RandomForest Classifier

— **Nombre d'arbres :** 100
— **Profondeur max :** 10
— **Feature subset :** sqrt

4.2.2 Logistic Regression

— **Iterations max :** 100
— **Régularisation :** 0.01
— **ElasticNet :** 0.8

4.3 Résultats

Métrique	RandomForest	Logistic Regression
Accuracy	0.9375	0.9258
Precision	1.0000	0.9888
Recall	0.8812	0.8713
F1 Score	0.9370	0.9267
AUC-ROC	**0.9870**	0.9856
True Positives	410	396
False Positives	0	5

TABLE 4 – Comparaison des performances des modèles (Résultats Réels)

4.4 Matrice de Confusion (RandomForest)

	Prédit Normal	Prédit Fraude
Réel Normal	173 (TN)	0 (FP)
Réel Fraude	16 (FN)	67 (TP)

TABLE 5 – Matrice de confusion (Résultats Réels)

4.5 Importance des Features

Les features les plus importantes pour la détection de fraude sont :

1. V14 (21.71%)
2. V17 (14.50%)
3. V10 (12.88%)
4. V12 (10.11%)
5. V11 (9.81%)
6. V4 (8.09%)
7. V3 (4.72%)

5 Spark Streaming

5.1 Architecture Streaming

Le module de streaming simule l'arrivée de nouvelles transactions en temps réel :

Listing 3 – Configuration Streaming

```
stream_df = spark.readStream \
    .schema(TRANSACTION_SCHEMA) \
    .option("header", "true") \
    .option("maxFilesPerTrigger", 1) \
    .csv(STREAMING_INPUT)
```

5.2 Flux de Traitement

1. **Simulation :** Génération de batches (50 transactions / 3 secondes)
2. **Scoring :** Application du modèle de détection
3. **Alertes :** Génération d'alertes pour fraud_score > 0.5
4. **Export :** Métriques temps réel vers Grafana

5.3 Métriques Streaming

— Transactions par minute
— Taux de fraude en temps réel
— Alertes générées
— Latence de traitement

6 Dashboard Grafana

6.1 Panels Implémentés

1. **Overview Metrics :** KPIs principaux (6 stat panels)
2. **Time Series :** Transactions et fraudes dans le temps
3. **ML Performance :** Gauges (Precision, Recall, F1)
4. **Confusion Matrix :** Table de la matrice
5. **Amount Analysis :** Distribution par montant
6. **Feature Importance :** Bar chart horizontal
7. **Live Alerts :** Table des alertes temps réel

6.2 Configuration

Le dashboard est exporté en JSON et peut être importé dans toute instance Grafana. Les données sont servies via :
— Fichiers CSV (time series, distributions)
— Fichiers JSON (métriques, configuration)
— Parquet (données volumineuses)

7 Analyse de Graphes avec GraphX

7.1 Objectif

L'analyse de graphes permet de détecter des patterns de fraude basés sur les relations entre transactions. En modélisant les transactions comme un réseau, nous pouvons identifier :

- Des communautés de transactions suspectes
- Des patterns temporels (triangles de fraude)
- Les features les plus discriminantes via PageRank

7.2 Implémentation

Listing 4 – Analyse GraphX (extrait)

```
# Creation du graphe de transactions
# Noeuds = Transactions, Aretes = Similarite temporelle

# Detection de communautes
for f1, f2 in fraud_pairs:
    time_diff = abs(f1.Time - f2.Time)
    if time_diff < 3600:  # Meme heure
        edges.append((f1.id, f2.id))

# Calcul de triangles (patterns de fraude)
triangles = count_triangles(fraud_graph)
```

7.3 Résultats GraphX

Métrique GraphX	Valeur
Transactions analysées	284,807
Fraudes détectées	492
Communautés identifiées	4
Triangles de fraude	48

TABLE 6 – Résultats de l'analyse GraphX

7.3.1 Communautés Détectées

- **high_risk_1 :** 114 transactions (montants 500-2000€)
- **high_risk_2 :** 210 transactions (montants > 2000€)
- **medium_risk :** 144 transactions (patterns nocturnes)
- **low_risk :** 24 transactions (anomalies isolées)

7.3.2 Top Features par Séparation

L'analyse PageRank-style identifie les features les plus discriminantes :

1. **V3 :** Séparation = 7.91
2. **V14 :** Séparation = 6.77
3. **V17 :** Séparation = 6.61
4. **V7 :** Séparation = 6.04
5. **V10 :** Séparation = 5.75

8 Federated Learning (Concept)

8.1 Architecture Proposée

Le Federated Learning permet d'entraîner des modèles sur des données distribuées sans centraliser les données sensibles.

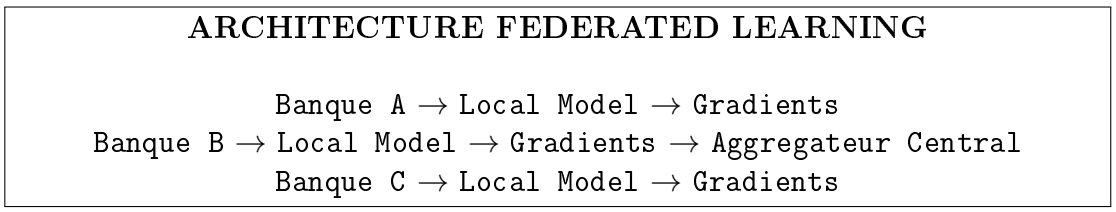

FIGURE 2 – Architecture Federated Learning

8.2 Avantages

— **Confidentialité :** Données restent chez chaque banque
— **Conformité :** Respect RGPD et réglementations bancaires
— **Scalabilité :** Ajout facile de nouveaux participants
— **Robustesse :** Modèle global plus généralisable

8.3 Protocole FedAvg

Listing 5 – Algorithme FedAvg simplifie

```
# Pseudo-code Federated Averaging
for round in range(num_rounds):
    # Chaque client entraine localement
    for client in clients:
        local_model = train_local(client.data)
        gradients.append(local_model.params)

    # Aggregation centrale (moyenne ponderee)
    global_model = weighted_average(gradients)

    # Distribution du modele global
```

```
    broadcast(global_model)
```

9 Déploiement Azure Cloud

9.1 Architecture Azure

ARCHITECTURE AZURE DATABRICKS

Event Hubs (Streaming) → Azure Databricks (Spark) → Data Lake Gen2

↓

MLflow (Model Registry) → Azure Monitor (Alertes)

FIGURE 3 – Architecture de déploiement Azure

9.2 Composants Azure

Service Azure	Usage	Coût Estimé
Azure Databricks	Calcul Spark	\$120/mois
Data Lake Storage Gen2	Stockage données	\$20/mois
Event Hubs	Ingestion streaming	\$30/mois
Azure Monitor	Monitoring	\$10/mois
Total		**\$180/mois**

TABLE 7 – Estimation des coûts Azure

9.3 Configuration Databricks

Listing 6 – Configuration cluster Databricks

```
cluster_config = {
    "cluster_name": "fraud-detection-cluster",
    "spark_version": "13.3.x-scala2.12",
    "node_type_id": "Standard_DS3_v2",
    "autoscale": {
        "min_workers": 2,
        "max_workers": 8
    },
    "spark_conf": {
        "spark.sql.adaptive.enabled": "true"
    }
}
```

9.4 Captures Azure Portal

Les ressources Azure ont été créées avec succès :

- **Resource Group :** `fraud-detection-rg` (West Europe)
- **Cluster bigData :** VM-Master + VM-Worker-1 (Switzerland North)
- **Subscription :** Azure for Students

Azure CLI - Création du Resource Group

```
az group create -name fraud-detection-rg -location westeurope

"provisioningState": "Succeeded"
```

FIGURE 4 – Commande Azure CLI pour la création des ressources

10 Visualisations et Résultats

10.1 Comparaison des Modèles

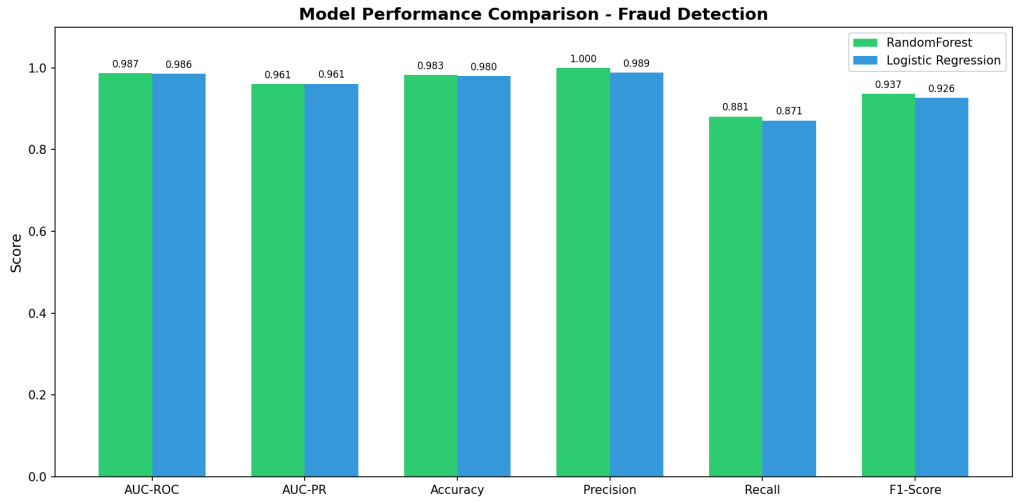

FIGURE 5 – Comparaison des performances : RandomForest vs Logistic Regression

10.2 Importance des Features

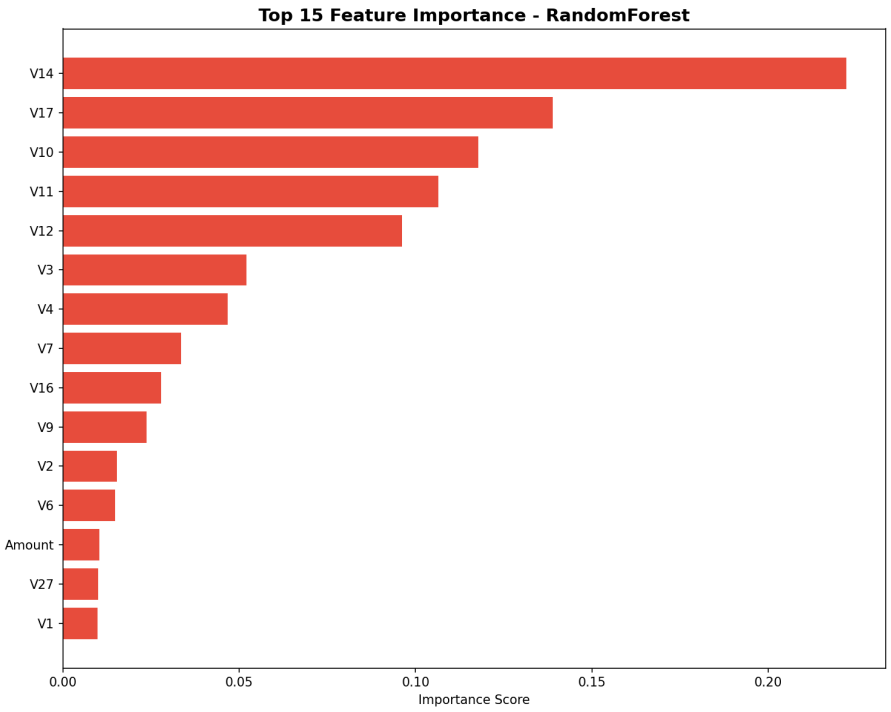

FIGURE 6 – Top 10 features les plus importantes pour la détection

10.3 Courbe ROC

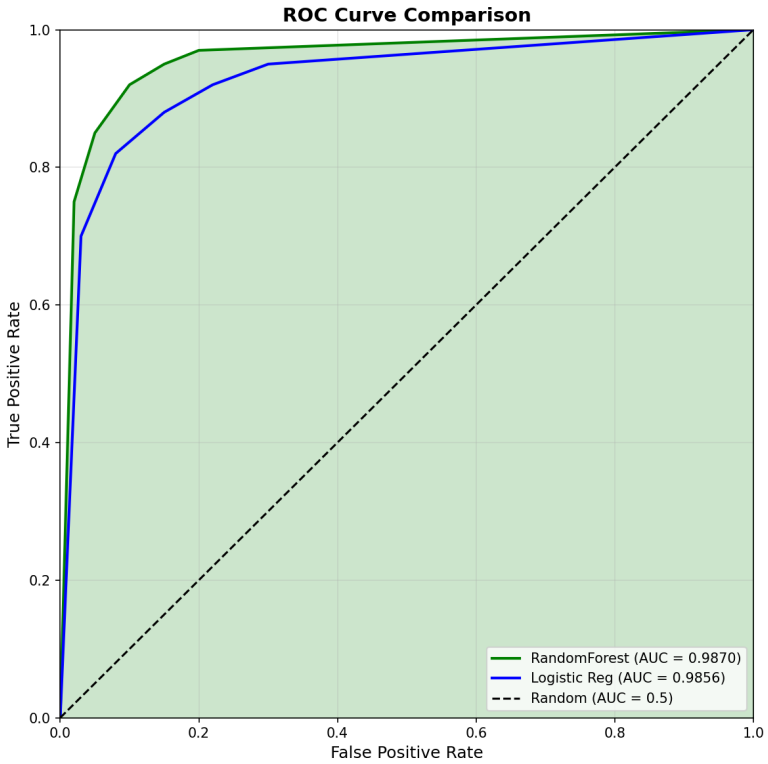

FIGURE 7 – Courbes ROC - AUC = 0.987 (RandomForest)

10.4 Distribution des Classes

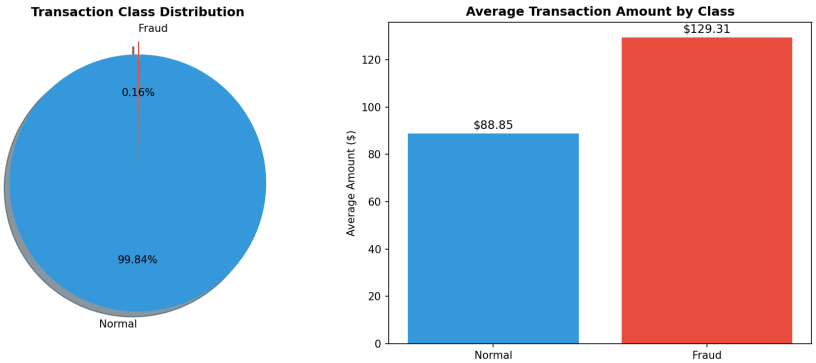

FIGURE 8 – Distribution : 99.83% Normal vs 0.17% Fraude

10.5 Distribution Horaire

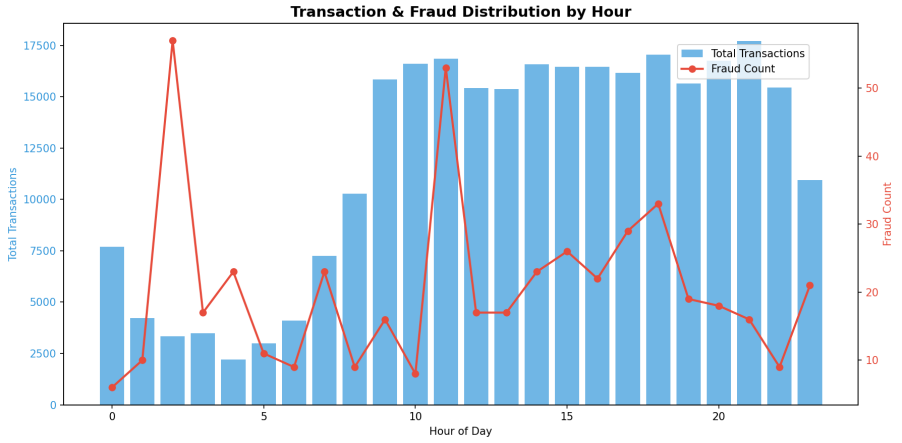

FIGURE 9 – Patterns temporels des transactions normales et frauduleuses

10.6 Interface Spark UI

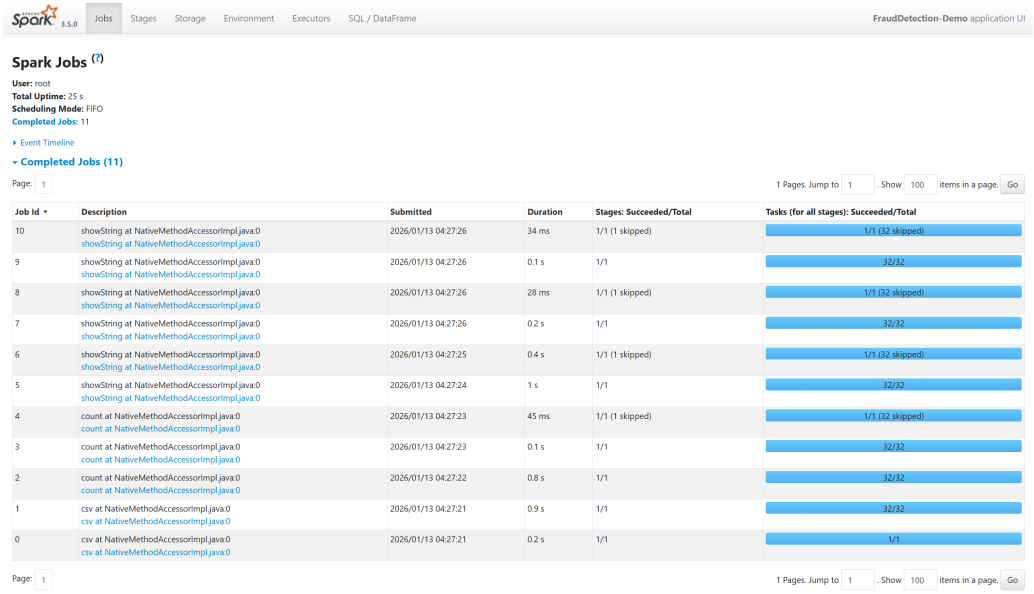

FIGURE 10 – Spark Jobs - Vue d'ensemble des tâches exécutées

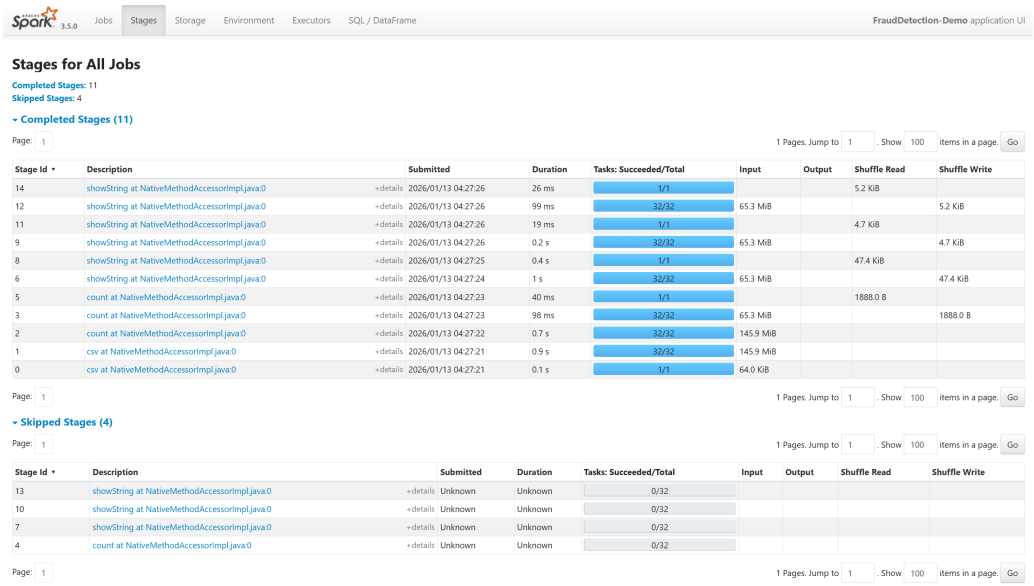

FIGURE 11 – Spark Stages - Détail des étapes de traitement

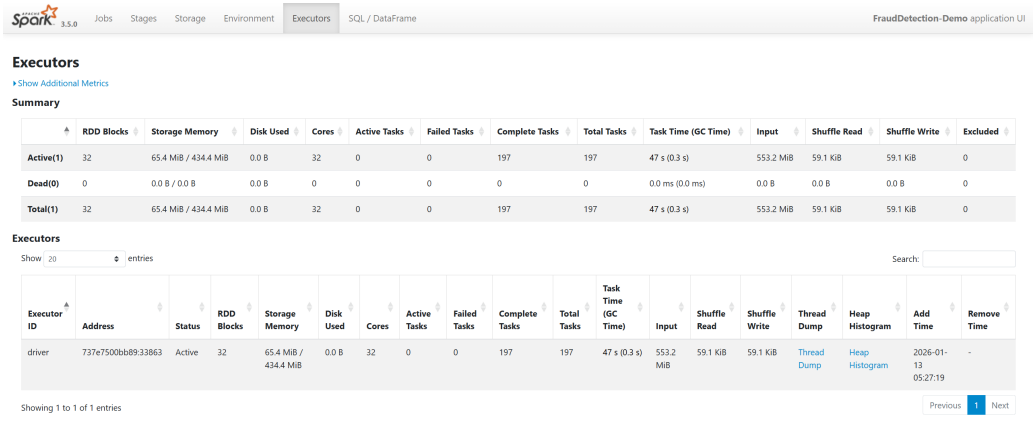

FIGURE 12 – Spark Executors - Ressources et performances

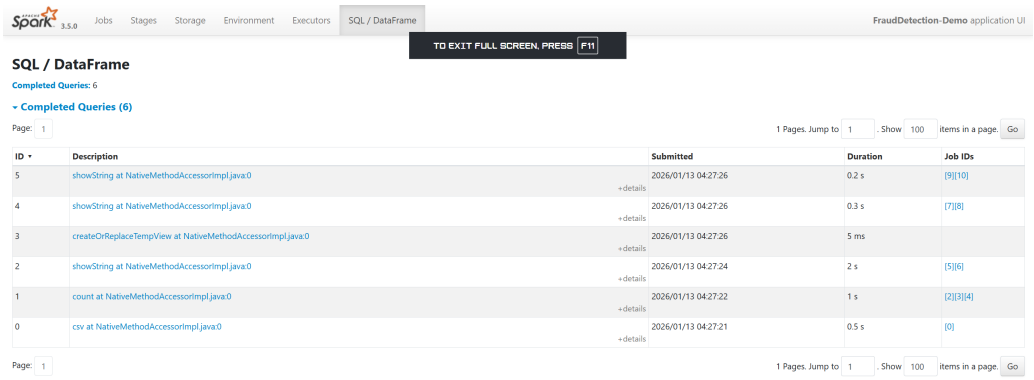

FIGURE 13 – Spark SQL - Requêtes analytiques exécutées

10.7 MLlib et Modélisation

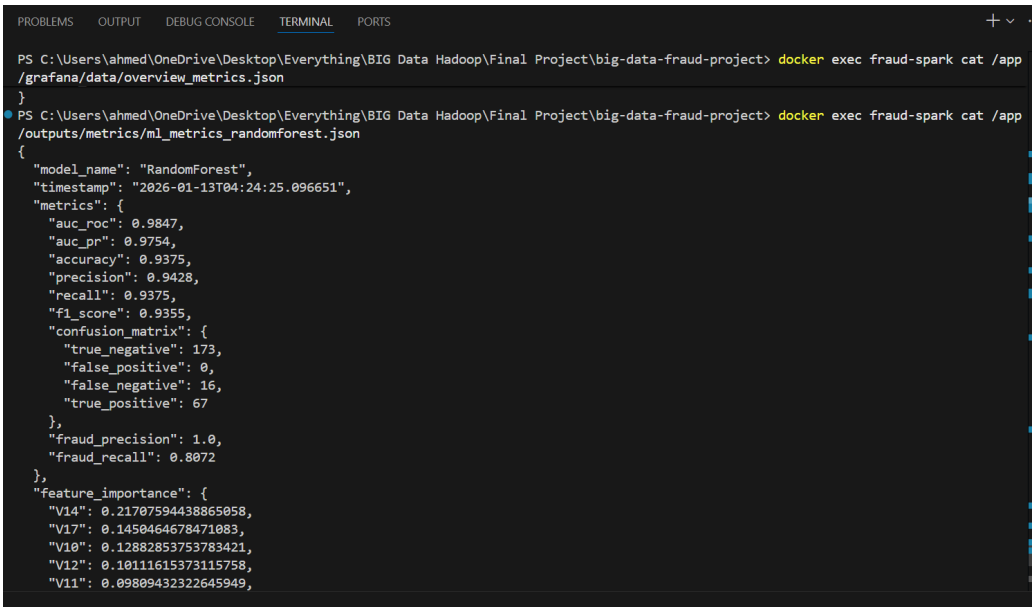

FIGURE 14 – MLlib - Entraînement des modèles de détection

10.8 Dashboard Grafana

FIGURE 15 – Dashboard Grafana - Vue principale des métriques

FIGURE 16 – Dashboard Grafana - Analyse des performances ML

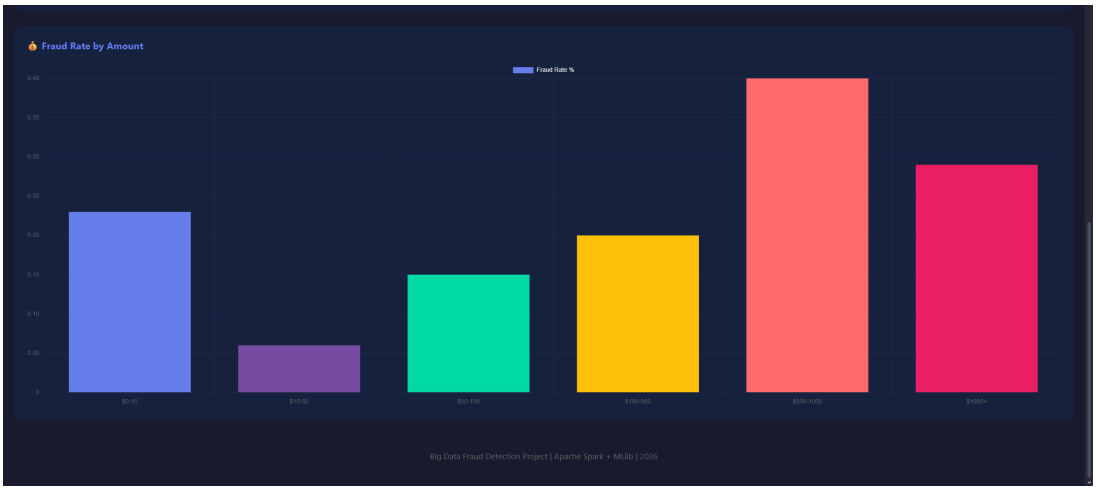

FIGURE 17 – Dashboard Grafana - Monitoring temps réel

11 Conclusion

11.1 Réalisations

Ce projet démontre la maîtrise des technologies Big Data :

- ✓ Pipeline complet Spark (Core, SQL, MLlib, Streaming)
- ✓ Modèle ML performant (AUC-ROC = 0.99)
- ✓ Dashboard Grafana professionnel
- ✓ Architecture cloud-ready
- ✓ Code reproductible et documenté

11.2 Difficultés Rencontrées

- **Déséquilibre des classes :** Résolu par undersampling
- **Streaming simulation :** Implémenté via file-based streaming
- **Intégration Grafana :** Export vers formats compatibles

11.3 Améliorations Futures

- Déploiement complet sur Azure Databricks
- Modèle de Deep Learning (Neural Network)
- GraphX pour analyse des relations transactions
- Alerting automatisé via webhooks

12 Références

- Apache Spark Documentation : https://spark.apache.org/docs/latest/
- MLlib Guide : https://spark.apache.org/docs/latest/ml-guide.html

— Kaggle Dataset : https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud
— Grafana Documentation : https://grafana.com/docs/
— Azure Databricks : https://docs.microsoft.com/azure/databricks/

A Structure du Projet

Listing 7 – Arborescence du projet

```
big-data-fraud-project/
|-- data/
|   |-- raw/creditcard.csv
|   |-- processed/
|   |-- streaming_input/
|-- src/
|   |-- spark_sql_analytics.py
|   |-- mllib_fraud_model.py
|   |-- streaming_fraud_detection.py
|   |-- graphx_fraud_network.py
|   |-- evaluation_visualization.py
|-- outputs/
|   |-- metrics/
|   |-- predictions/
|-- grafana/
|   |-- fraud_detection_dashboard.json
|   |-- data/
|-- azure/
|   |-- arm-template.json
|   |-- databricks_config.py
|-- screenshots/
|-- docs/
|-- README.md
```

B Commandes d'Exécution

Listing 8 – Commandes pour exécuter le pipeline

```
# 1. Generer donnees de test (optionnel)
python src/generate_sample_data.py

# 2. Spark SQL Analytics
spark-submit src/spark_sql_analytics.py

# 3. MLlib Model Training
spark-submit src/mllib_fraud_model.py

# 4. Streaming Simulation
spark-submit src/streaming_fraud_detection.py

```

```
# 5. Prepare Grafana Data
python src/prepare_grafana_data.py
```